Tuning Your Code With the Performance Analyzer

By John Schroeder

With Visual PRO/5® Rev 4.0 and BBj® Rev 2.01, BASIS released a new utility designed to aid in analyzing your application code for performance bottlenecks. This utility makes use of a new feature of the SETTRACE verb, which allows you to turn on a timer to get millisecond accuracy on timing each statement that is executed during the trace. The trace output is directed to a file, and this file is fed into the new Performance Analyzer (**_prof.bbx** or **_prof.bbj**) utility.

Please note that while the examples shown in this article use BBj, you can also use Visual PRO/5 to look at a trace file from a PRO/5 or Visual PRO/5 system. When trace output is directed to a file, the default setting is to include the timing information. The default for a trace to the screen is not to include the timing information. To turn off the timing information when tracing to a file, you can use the following MODE on your SETTRACE statement:

```
SETTRACE(chan,MODE="UNTIMED")
```

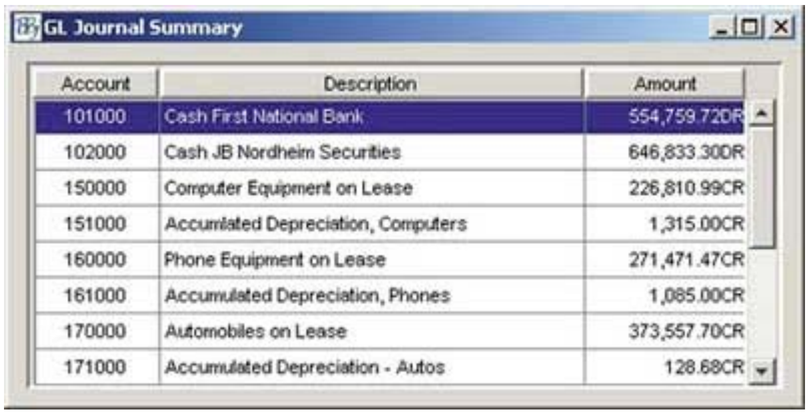
You can turn on the timing information with the mode "TIMED."

Using the timing information in the trace file, the Performance Analyzer (PA) builds a table showing the number of times each line of code was executed. This table also shows the total elapsed time and the average elapsed time per line, as well as the percentage of the duration of the run taken up by each line of code. This useful information can assist you in isolating and removing bottlenecks in your application. Additionally, a summary page shows the overall timings for each program in the trace run.

There are two programs available from the Cool Stuff section of the BASIS Web page, www.basis.com/devtools/coolstuff, and in the enclosed Advantage CD, which illustrate the use of the Performance Analyzer. These are **perfSample.bbj** and **perfSample1.bbj**, along with supporting files. The first program is the original program, which was run through the PA to see where improvements could be made. The second is a restructured version of the first, which reduces a bottleneck in the original, and improves overall performance by about 50%.

perfSample.bbj reads a journal entry detail file in general ledger account number sequence. Then it summarizes the postings for each account. The data is displayed in a grid, as shown in Figure 1.

Figure 1. Journal summary display.



GL Journal Summary

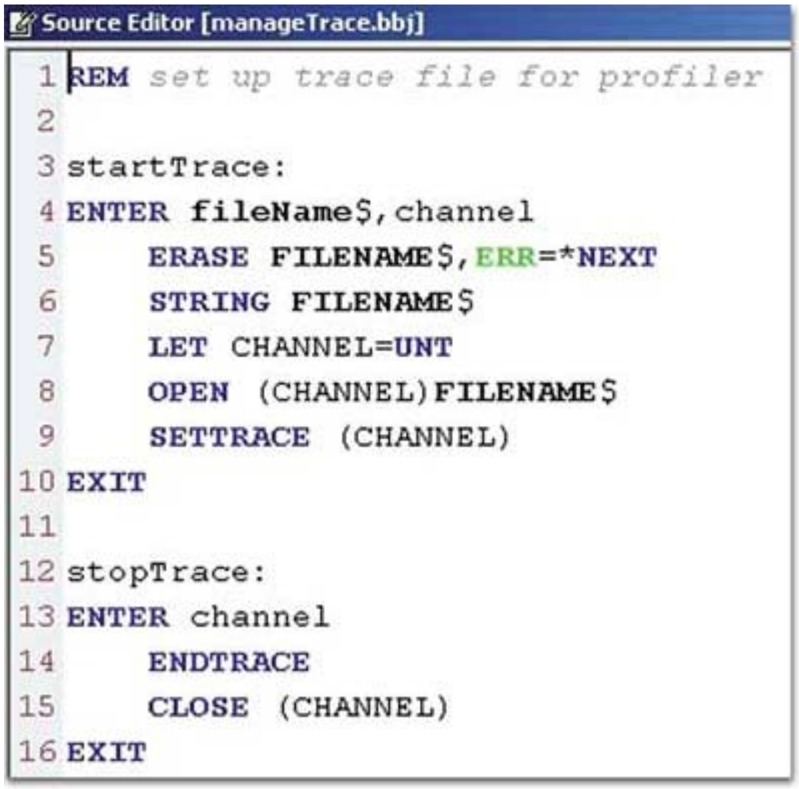
Account	Description	Amount
101000	Cash First National Bank	554,759.72DR
102000	Cash JB Nordheim Securities	646,833.30DR
150000	Computer Equipment on Lease	226,810.99CR
151000	Accumlated Depreciation, Computers	1,315.00CR
160000	Phone Equipment on Lease	271,471.47CR
161000	Accumulated Depreciation, Phones	1,085.00CR
170000	Automobiles on Lease	373,557.70CR
171000	Accumulated Depreciation - Autos	128.68CR

Let's assume that we plan to move this program to an environment where the journal files are very large.

We want to be sure it is as efficient as possible, so we will use the PA to determine if there are areas of code where the performance can be improved.

Let's set up the program **perfSample.bbj**, so it can be used with the PA. To do this we need to start a trace to a file while the program is running and then turn off the trace. The program, **manageTrace.bbj** (**Figure 2**), can be used to create a file and start/stop the trace.

Figure 2. manageTrace.bbj.

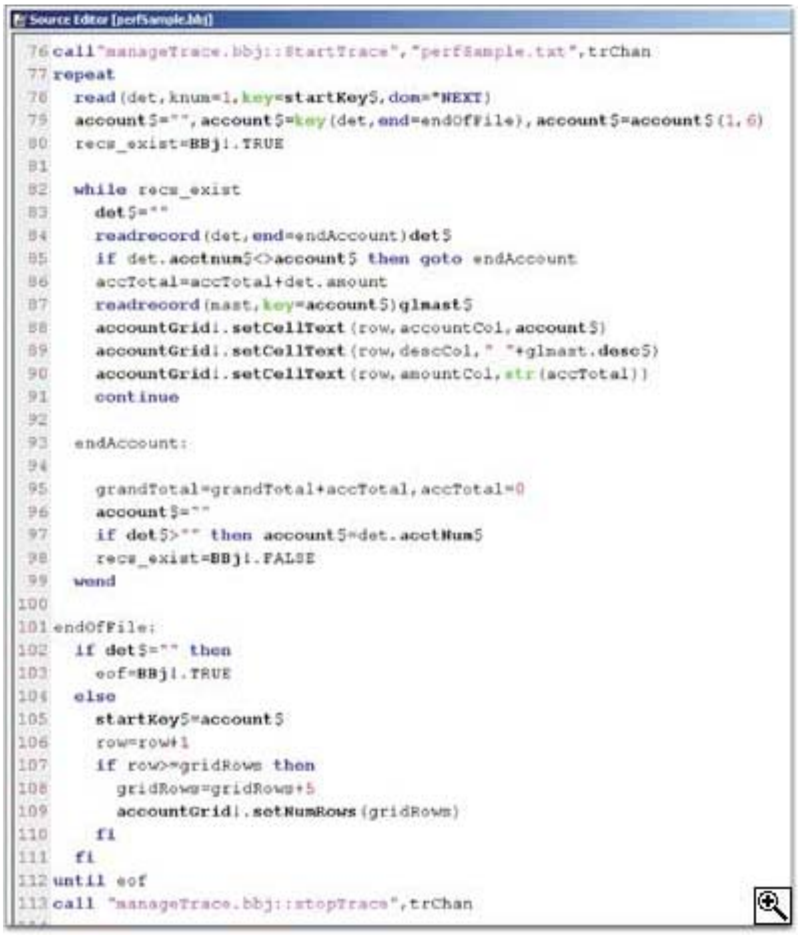


```
REM set up trace file for profiler

startTrace:
ENTER fileName$,channel
      ERASE FILENAME$,ERR=*NEXT
      STRING FILENAME$
      LET CHANNEL=UNT
      OPEN (CHANNEL)FILENAME$
      SETTRACE (CHANNEL)
EXIT

stopTrace:
ENTER channel
      ENDTRACE
      CLOSE (CHANNEL)
EXIT
```

This program can be called with one of two labels, startTrace or stopTrace. The procedure is to call it to start the trace at a point near the suspected bottleneck, and to end the trace after that process is finished.

The segment from perfSample.bbj illustrates the use of the manageTrace.bbj public program.

Rather than put the file and trace management code into the application, it may be easier to use a program like **manageTrace.bbj** to handle starting and stopping the trace.



```
call"manageTrace.bbj::StartTrace","perfSample.txt",trChan
repeat
  read(det,knum=1,key=startKey$,dom=*NEXT)
  account$="",account$=key(det,end=endOfFile),account$=account$(1,6)
  recs_exist=BBj!.TRUE

  while recs_exist
    det$=""
    readrecord(det,end=endAccount)det$
    if det.acctnum$<>account$ then goto endAccount
    accTotal=accTotal+det.amount
    readrecord(mast,key=account$)glmast$
    accountGrid!.setCellText(row,accountCol,account$)
    accountGrid!.setCellText(row,descCol," "+glmast.desc$)
    accountGrid!.setCellText(row,amountCol,str(accTotal))
    continue

  endAccount:

    grandTotal=grandTotal+accTotal,accTotal=0
    account$=""
    if det$>"" then account$=det.acctNum$
    recs_exist=BBj!.FALSE
  wend

endOfFile:
  if det$="" then
    eof=BBj!.TRUE
  else
    startKey$=account$
    row=row+1
    if row>=gridRows then
      gridRows=gridRows+5
      accountGrid!.setNumRows(gridRows)
    fi
  fi
until eof
call "manageTrace.bbj::stopTrace",trChan
```

Figure 3. perfSample.bbj with calls to manageTrace.bbj.

Figure 3 shows the code from **perfSample.bbj** with the calls to **manageTrace.bbj** encompassing the code that reads in the journal detail information and loads the output grid.

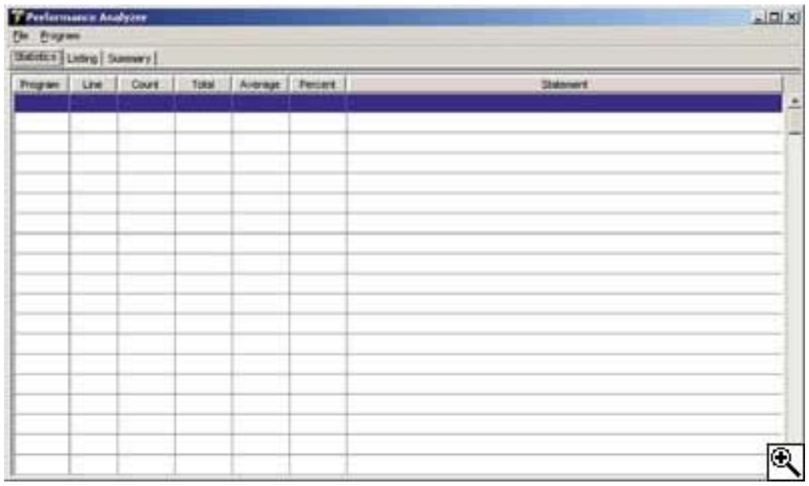
When we run the program we get a trace file, which looks like this:

```
[11] EXIT
>EXITING TO: perfSample.bbj
    {0.54}{0.52}
[78] REPEAT
    {0.72}{0.18}
[79] READ (det,KNUM=1,KEY=startKey$,DOM=*NEXT)
    {1.21}{0.48}
[80] LET account$="",account$=key(det,END=endOfFile),account$=account$(1,6)
    {1.97}{0.76}
[81] LET recs_exist=BBj!.TRUE
>JAVA: com.basis.bbj.proxies.BBjProxy.TRUE returns int
    {2.46}{0.48}
[83] WHILE recs_exist
    {2.67}{0.2}
[84] LET DET$=""
    {2.84}{0.17}
[85] READ RECORD (det,END=endAccount)det$
    {3.12}{0.28}
```

The number in the square brackets at the beginning of the program line is the relative line number in the program file. The programs run and called in this example are unnumbered BBj programs. If they were numbered, the program line numbers would be listed instead of the relative numbers. The major difference between this trace and an ordinary trace is the timing information that appears after each line. The numbers in the braces show the total elapsed time and the elapsed time used in executing the preceding statement. All times are given in milliseconds.

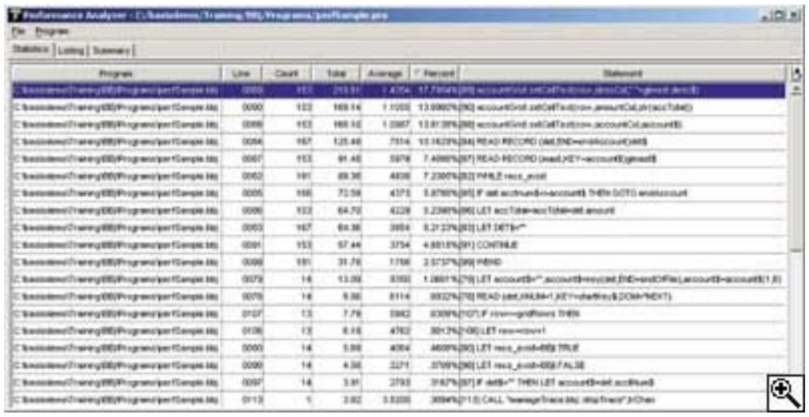
Now that we've run the program and saved the trace output, it is time to use the PA. The main PA screen is illustrated in **Figure 4**.

Figure 4. Main Performance Analyzer window.



Using the File|Open menu, we select a text file that has been created by the trace run. When we open the text file, the PA summarizes the data into the grid. The data can be sorted by any of the columns in the grid. Initially, we will sort by the **Percent** column so that the lines appear in descending order, by the percentage of the total elapsed time. Using the text file created in the trace run described in **Figure 4** and sorting by descending percentage, we get the results shown in **Figure 5**.

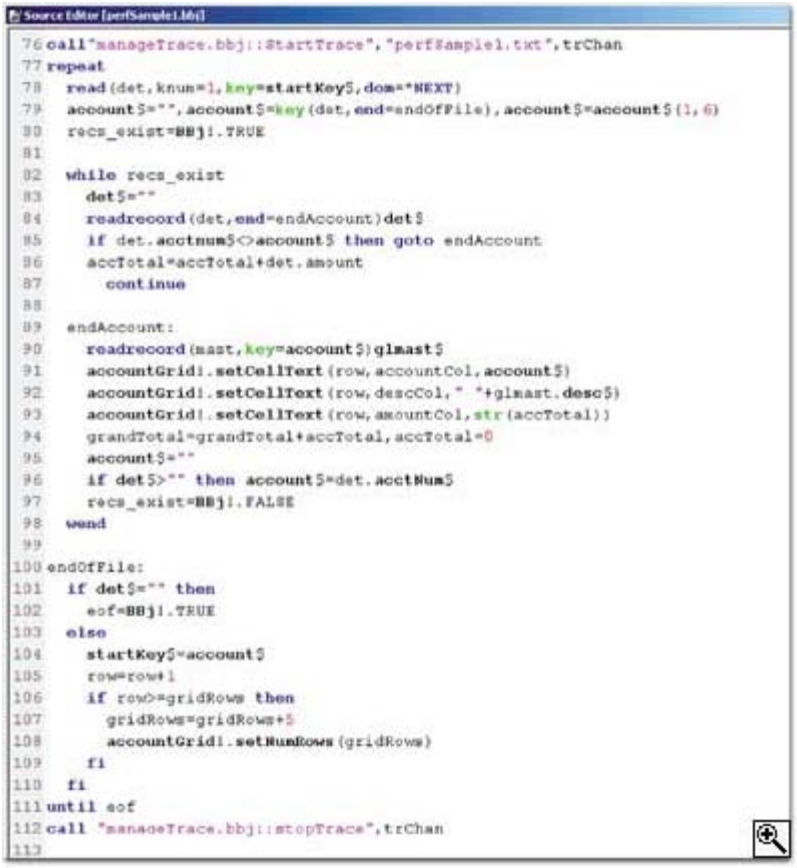
Figure 5. Performance Analyzer result table for perfSample.bbj.


Performance Analyzer - C:\basisdemo\Training\BBj\Programs\perfSample.pro
File
Program
Statistics
Listing
Summary

Program	Line	Count	Total	Average	Percent	Statement
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0089	153	219.51	1.4054	17.7954%	[89] accountGrid.setCellText(row,descCol,""+glmast.desc$)
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0090	122	169.14	1.1020	13.0080%	[90] accountGrid.setCellText(row,amountCol,str(accTotal))
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0088	153	168.10	1.0987	13.8139%	[88] accountGrid.setCellText(row,accountCol,account$)
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0084	167	125.46	.7514	10.1623%	[84] READ RECORD (det,END=endAccount)det$
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0087	153	91.46	.5978	7.4088%	[87] READ RECORD (mast,KEY=account$)glmast$
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0082	181	89.36	.4938	7.2300%	[82] WHILE recs_exist
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0085	166	72.58	.4373	5.8780%	[85] IF det.acctnum$<>account$ THEN GOTO endAccount
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0086	153	64.70	.4228	5.2398%	[86] LET accTotal=accTotal+det.amount
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0083	167	64.36	.3854	5.2123%	[83] LET DET$=""
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0091	153	57.44	.3754	4.6518%	[91] CONTINUE
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0099	181	31.79	.1756	2.5737%	[99] WEND
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0079	14	13.09	.9350	1.0601%	[79] LET account$="",account$=key(det,END=endOfFile),account$=account$(1,8)
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0076	14	8.56	.6114	.6932%	[76] READ (det,KNUM=1,KEY=startKey$,DOM=NEXT)
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0107	13	7.79	.5992	.6309%	[107] IF row=gridRows THEN
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0108	13	6.19	.4762	.5013%	[108] LET row=row+1
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0080	14	5.68	.4054	.4600%	[80] LET recs_exist=BBj.TRUE
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0098	14	4.58	.3271	.3709%	[98] LET recs_exist=BBj.FALSE
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0097	14	3.91	.2793	.3167%	[97] IF det$<>"" THEN LET account$=det.acctNum$
C:\basisdemo\Training\BBj\Programs\perfSample.bbj	0113	1	3.82	3.8200	.3094%	[113] CALL "manageTrace.bbj::stopTrace",hChan

The table in **Figure 5** was generated by the PA, which read the data from the trace file and summarized it in the table. Depending on the size of the trace, this can be a lengthy process. You will probably need to view the data for a given run more than once. To make this easier for you we have included a **Save** option on the File menu. This allows you to save the summary information into a file with the extension .pro, as shown in the title bar in **Figure 5.** This file is an MKEYED file and can be read into the grid very quickly for review.

The result table in Figure 5 shows that a lot of time was taken up in reading the data from the detail file, reading the description from the master file, and moving the data into the grid. We cannot eliminate this code. Each line is necessary for the program to function properly. However, we can reorganize the code to make it more efficient. If we look at the innermost while/wend loop, we see that the lines that read the GL Master file and those that move the account info into the grid are in this loop. Since the account information itself does not change in this loop, it can be moved outside the loop. Also, the total is accumulated as the detail records are read, so it isn't necessary to continually update the grid with the total until all the records for a particular account have been read in and totaled. Let's move these lines to the outermost repeat/until loop as shown in **Figure 6**.



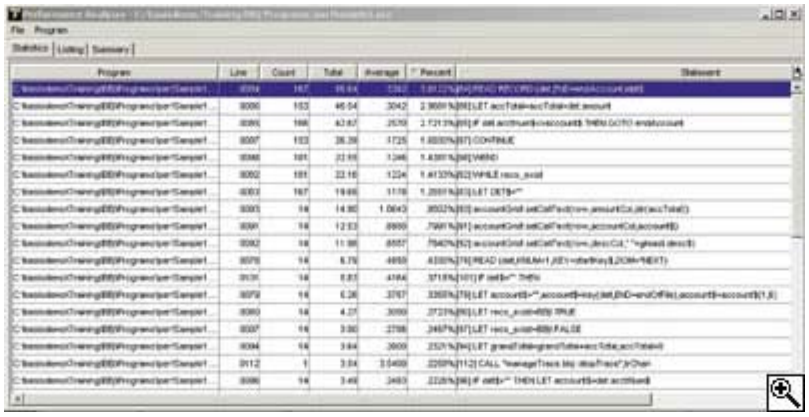
Source Editor [perfSample1.bbj]

```
call"manageTrace.bbj::StartTrace","perfSample1.txt",trChan
repeat
  read(det,knum=1,key=startKey$,dom=*NEXT)
  account$="",account$=key(det,end=endOfFile),account$=account$(1,6)
  recs_exist=BBj!.TRUE

  while recs_exist
    det$=""
    readrecord(det,end=endAccount)det$
    if det.acctnum$<>account$ then goto endAccount
    accTotal=accTotal+det.amount
      continue

  endAccount:
    readrecord(mast,key=account$)glmast$
    accountGrid!.setCellText(row,accountCol,account$)
    accountGrid!.setCellText(row,descCol," "+glmast.desc$)
    accountGrid!.setCellText(row,amountCol,str(accTotal))
    grandTotal=grandTotal+accTotal,accTotal=0
    account$=""
    if det$>"" then account$=det.acctNum$
    recs_exist=BBj!.FALSE
  wend

endOfFile:
  if det$="" then
    eof=BBj!.TRUE
  else
    startKey$=account$
    row=row+1
    if row>=gridRows then
      gridRows=gridRows+5
      accountGrid!.setNumRows(gridRows)
    fi
  fi
until eof
call "manageTrace.bbj::stopTrace",trChan
```

Figure 6. Move these lines to the outermost repeat/until loop.

Now run the program again and analyze the trace output to see if there is any improvement. The results are shown in **Figure 7**.

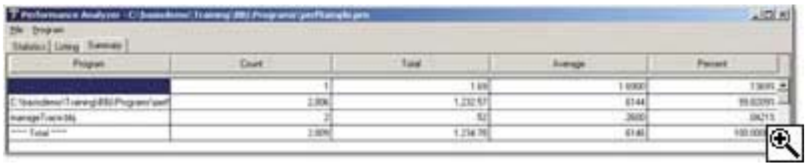
Figure 7. Performance Analyzer result table for second trace run.



Program	Line	Count	Total	Average	Percent	Statement
C:\basisdemo\Training\BBj\Programs\perfSample1...	0084	167	95.64	.5362	[illegible]	[84] READ RECORD (det,END=endAccount)det$
C:\basisdemo\Training\BBj\Programs\perfSample1...	0086	153	46.54	.3042	2.9681%	[86] LET accTotal=accTotal+det.amount
C:\basisdemo\Training\BBj\Programs\perfSample1...	0085	166	42.67	.2570	2.7213%	[85] IF det.acctnum$<>account$ THEN GOTO endAccount
C:\basisdemo\Training\BBj\Programs\perfSample1...	0087	153	26.39	.1725	1.8830%	[87] CONTINUE
C:\basisdemo\Training\BBj\Programs\perfSample1...	0098	181	22.55	.1246	1.4381%	[98] WEND
C:\basisdemo\Training\BBj\Programs\perfSample1...	0082	181	22.16	.1224	1.4132%	[82] WHILE recs_exist
C:\basisdemo\Training\BBj\Programs\perfSample1...	0083	167	19.66	.1178	1.2591%	[83] LET DET$=""
C:\basisdemo\Training\BBj\Programs\perfSample1...	0093	14	14.90	1.0643	.9532%	[93] accountGrid!.setCellText(row,amountCol,str(accTotal))
C:\basisdemo\Training\BBj\Programs\perfSample1...	0091	14	12.53	.8950	.7991%	[91] accountGrid!.setCellText(row,accountCol,account$)
C:\basisdemo\Training\BBj\Programs\perfSample1...	0092	14	11.98	.8557	.7640%	[92] accountGrid!.setCellText(row,descCol," "+glmast.desc$)
C:\basisdemo\Training\BBj\Programs\perfSample1...	0078	14	6.79	.4850	.4330%	[78] READ (det,KNUM=1,KEY=startKey$,DOM=NEXT)
C:\basisdemo\Training\BBj\Programs\perfSample1...	0101	14	5.83	.4164	.3718%	[101] IF det$="" THEN
C:\basisdemo\Training\BBj\Programs\perfSample1...	0079	14	5.28	.3767	.3368%	[79] LET account$="",account$=key(det,END=endOfFile),account$=account$(1,6)
C:\basisdemo\Training\BBj\Programs\perfSample1...	0080	14	4.27	.3050	.2723%	[80] LET recs_exist=BBj!.TRUE
C:\basisdemo\Training\BBj\Programs\perfSample1...	0097	14	3.90	.2786	.2487%	[97] LET recs_exist=BBj!.FALSE
C:\basisdemo\Training\BBj\Programs\perfSample1...	0094	14	3.64	.2600	.2321%	[94] LET grandTotal=grandTotal+accTotal,accTotal=0
C:\basisdemo\Training\BBj\Programs\perfSample1...	0112	1	3.04	3.0400	.2202%	[112] CALL "manageTrace.bbj::stopTrace",trChan
C:\basisdemo\Training\BBj\Programs\perfSample1...	0096	14	3.49	.2493	.2226%	[96] IF det$>"" THEN LET account$=det.acctNum$

The results show that the lines, which move the data into the grid, while taking approximately the same amount of time per line as in the previous case, are now executed only 14 times, instead of 153 times, as was true previously. Additionally, the READ on the master file has fallen off the first page of the chart, indicating that its contribution is significantly smaller. We can confirm this by looking at the summary tabs for each run. These are shown in **Figure 8**.

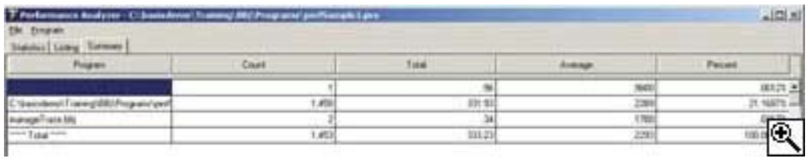
Figure 8. Summary table for first run.



Program	Count	Total	Average	Percent
	1	1.69	1.6900	.1369%
C:\basisdemo\Training\BBj\Programs\perf	2,006	1,232.57	.6144	99.8299%
manageTrace.bbj	2	52	.2600	.0421%
---- Total ----	2,009	1,234.78	.6146	100.000

As **Figure 8** shows, the total time for the first run was 801 milliseconds. The second run, shown in **Figure 9** presents a total time of 337 milliseconds for a savings of more than 50%.

Figure 9. Summary table for second run.



Program	Count	Total	Average	Percent
	1	96	9600	[illegible]
C:\basisdemo\Training\BBj\Programs\perf	1,450	[illegible]	2289	[illegible]
manageTrans.bbj	2	34	1700	[illegible]
**** Total ****	1,453	[illegible]	2293	[illegible]

Guided by the Performance Analyzer, we have reorganized the code to run more efficiently. It's easy to see that using the new Performance Analyzer can help isolate performance bottlenecks and enable you to tune your application code for best performance.

The code in this article is available on the Advantage CD and also on our website at:
www.basis.com/devtools/coolstuff/index.html